REMARKS

Claims 1-31 were pending and rejected. Claims 1, 24, 30 and 31 are being amended. Claims 6 and 11-23 have been canceled. Claims 1-5, 7-10 and 24-31 remain pending. Reconsideration is respectfully requested.

In section 2, the Examiner rejected claims 22, 24, 30 and 31 under 35 USC § 112 as indefinite. Specifically, the Examiner rejected claim 22 and 24 for antecedent basis problems. The Examiner rejected claims 30 and 31 as including language considered unclear. Applicant has canceled claim 22. Applicant is amending claims 24, 30 and 31 to address the Examiner's concerns.

In section 3-6, the Examiner rejected claims 1-15 and 17-31 under 35 USC § 101 as directed to nonstatutory subject matter. Specifically, the Examiner rejected claims 1-10 and 17-31 as being directed to purely algorithmic processes. Applicant amended claim 1 and canceled claims 11-23, which as written suggested claim 1 was directed to pure software. As amended, remaining claims 1-5, 7-10 and 24-31 are no longer purely algorithmic. The Examiner rejected claims 11-15 as being directed to intangible subject matter. As stated above, Applicant has canceled claims 11-15.

Generally, in all § 102 and § 103 rejections discussed below, the prior art applied by the Examiner refers to software-based schemes. As stated above, these schemes are simplistic and do not yield performance comparable to hardware emulation. This is because, in most cases, there is no mechanism to convey execution-time information to software. For example, it is impossible for hardware to convey dynamic branch prediction information to software that might enable the software to generate different emulation sequences. Even if conveying dynamic information were possible, a special software routine would need to be invoked to regenerate a new code sequence – something that can be prohibitively expensive compared to a pure hardware solution. The hardware requirement is required by claims 1 and 24. Specifically, claim 1 as amended requires "<u>changing the computer system</u> dynamically for producing different dynamic execution information in response to said first dynamic execution information."

[emphasis added] Claim 24 as amended requires "means for responding to the dynamic execution information and for changing the computer system dynamically so that at least some dynamic execution information obtained on subsequent execution of the emulated sequence of instructions would be changed." [emphasis added]

Also, in all § 102 and § 103 rejections discussed below, the prior art cited by the Examiner involves re-compilation based on profile statistics or link time optimization. These prior art techniques are "offline" and not feasible during program execution for performance, storage and even code availability reasons. Further, these systems require advance knowledge about the possible target system, which may not always be available. At link-time, there may not be enough information about the actual hardware platform on which the program is going to execute to make such decisions. For example, in Wall, without knowledge about the actual number of registers present in the system where the program is going to run, link-time register allocation is impossible. Again, dynamic techniques are required by claims 1 and 24. Specifically, claim 1 as amended requires "changing the computer system dynamically for producing different dynamic execution information in response to said first dynamic execution information." [emphasis added] Claim 24 as amended requires "means for responding to the dynamic execution information and for changing the computer system dynamically so that at least some dynamic execution information obtained on subsequent execution of the emulated sequence of instructions would be changed." [emphasis added]

In sections 7 and 8, the Examiner rejected claims 1-3, 6, 10-13, 17, 18, and 22 under 35 USC § 102 as unpatentable over Altman. Claims 6 and 11-23 have been canceled. Accordingly, this rejection now applies only to claims 1-3 and 10.

Altman refers to software programs that translate one program to another. The Examiner (e.g., in comments 8.7 and 8.8) refers to the section entitled, "Three Types of Translation." The first line of that section begins, "Software-based binary translation...." The constraints, requirements, difficulties and design complexities as well as the performance of software-based schemes are substantially different from the claimed hardware emulation. Most software-based schemes cannot dynamically change instruction parameters. This is emphasized by Altman

under the section entitled "Dynamic Optimization", page 41, bullet 3 "Code Specialization," which mentions specialization "based on the invariance of parameter values." The claimed embodiments can handle changing or varying parameter values based on execution information.

Altman refers to several works of prior art including "Native Binary Translation" by Bala et al. It should be noted that Bala is also a software implementation that is much slower than the hardware emulation of the present invention. The design complexities and implementation details are substantially different from the claimed hardware embodiments. A difference is noted in the following instruction:

```
ADD R1, R2                    // R1 + R2  -> R1
```

Registers R1 and R2 are added and the result is placed in R1. The current state-of-the-art can remap the ADD instruction as follows:

```
ADD R1, R2, R3                   // R1 + R2  -> R3
```

In this case, rather than reusing R1, the destination is remapped to R3. This is done dynamically at run-time. Each time this instruction is executed, the emulation parameter R3 will be the destination.

The claimed embodiments go beyond this. Some embodiments of the present invention provides a hardware mechanism choosing different emulation parameters, in this case the register destinations, when it comes across the same or different ADD instructions. So, based on run-time information, one claimed embodiment may generate

```
ADD R1, R2, R3                   // R1 + R2  -> R3
```

the first time, while generating the following

```
ADD R1, R2, R10                   // R1 + R2  -> R10
```

the next time. Note that this different instruction that has been generated could have been for the very same ADD instruction (say, in a loop) in the original sequence or a completely different ADD instruction in the original sequence.

There is no mention in Altman beyond the original remap of the ISA, in our example, to target register R3. Specifically, there is no mention of the ability and value to generate different remappings, e.g., initially R3 and subsequently R10 in the above example.

Consider another example, this time involving dynamic branch prediction. The original instruction

```
BT/U #disp     // Branch for true condition, unlikely
```

would normally be emulated as

```
BNE/U R19, R63, TR0 // Branch for true condition, unlikely
```

However, in select embodiments of the present invention, based on dynamic branch prediction information that indicates that the branch is likely, a different emulation instruction is generated as follows:

```
BNE/L R19, R63, TR0 // Branch for true condition, LIKELY
```

Note that the new emulation instruction differs from the original instruction in semantics and behavior.

Further, Altman does not mention providing both the original and emulation instruction sequences. Para 3 just has a definition of translation.

Altman mentions basic block reordering only; there is no mention of changing instruction sequences. The basic block reordering reorders groups (blocks) of instructions to improve performance. There is no mention of dynamically changing the emulation sequence.

The requirements and constraints of a software-based binary translator are substantially different from hardware. In the software-based scheme, the original program is translated into a new binary program. This doubles the storage requirement on the computer system and that may not always be possible. Further, the computer executes the translated program only – it does not use the original program in any way. Translated programs are frequently much bigger and this increased footprint necessitates more storage and has adverse cache effects. Embodiments of the invention deal only with the original binary and generates the emulation sequence at run-time.

In sections 9 and 10, the Examiner rejected claim 4 under 35 USC § 103 as obvious over Altman in view of Kelly. Since the prior art has dealt with software-based schemes, it is not clear if or how the dynamic, run-time information is used in a hardware emulator. Embodiments of the invention explicitly talk about hardware needed in the instruction decode phase that generates the emulation instruction sequence prior to execution.

Kelly (C11, L54-67) talks about software that performs the translation and caches (or stores) the translated instructions. This has all the disadvantages of performance and storage explained earlier. Further, replacement of multiple condition codes is possible only with execution-time information. Note that Altman is a "survey" paper that has described several software-based approaches to solving this problem. However, they all share this significant disadvantage of not having execution time information available or the ability to use such information even if available. Even the "Dynamic Optimization" mentioned by Altman needs a dynamic translator which makes execution slow. This is mentioned in the very first line under that section.

Kelly (C14, L3-60) explains how a single instruction in the original sequence is emulated by a sequence of instructions in the native instruction set. As the Examiner has pointed out, each instruction in the emulation sequence can produce condition codes. However, only one of those

codes is effective and used. For example, if there is a check for overflow, only one code can be checked.

A claimed embodiment of the invention is different in how multiple codes can be produced and used. As explained in the original application, consider a piece of error-handling code that is first called in response to a large number of errors. In this case, even if a single type of error was highly unlikely, the pooled response for a large number of errors could mean that the error-handling code is likely to be called. So while the branch to any specific error code is unlikely, branch to the shared error handler is likely. Based on dynamic information, we see the following:

```
BT/U #disp      // Branch for true condition, unlikely
```

is translated as:

```
PT/L #disp, TR0      // Prepare target, prefetch, LIKELY
BNE R19, R63, TR0   // Branch for true condition, UNLIKELY
```

Note that the original condition code has yielded 2 different condition codes in the emulation sequence based on dynamic information. This is different from Kelly where multiple, but useless condition codes are produced as a byproduct of having executed multiple instructions in the emulation sequence.

Kelly, in the figure just below the abstract on page 1, shows how much of the translation is software based. As pointed out earlier, Kelly does not mention, let alone describe hardware emulation and, specifically, the claimed embodiment of dynamically changing instruction emulation parameters. Nowhere in Kelly is dynamic variation of instruction parameters mentioned.

In section 11, the Examiner rejected claims 5, 14 and 19 under 35 USC § 103 as obvious over Altman in view of Lethin. Claims 14 and 19 have been canceled. Accordingly, this rejection is applied only to claim 5. Lethin also employs a software-based translation scheme. Lethin has all the characteristics described in Kelly that have already been addressed. Lethin (P0381 and 0384) describes register allocation algorithms used in compilers as being multi-step

process. Embodiments of the invention are in a completely different context and relate to using execution time information collected from recursive calculations. The limitation of template-based translation schemes is that the emulation sequence is based on a static template and has little or no dynamic information. Embodiments of the invention avoid this limitation. Not only can our template be created at run-time, it can also be changed based on execution behavior.

In section 12, the Examiner rejected claims 7, 15 and 20 under 35 USC § 103 as obvious over Atman in view of Conte. Claims 15 and 20 have been canceled. Accordingly, this rejection is applied only to claim 7. Conte uses branch prediction hardware to generate profiling statistics which Conte claims is faster than using profiling software. These profile statistics are then used by the compiler to generate more optimized code. Conte is about speeding up the collection of profile statistics. Our invention is in a different field.

Sections 2 and 3 of Conte discuss the use of branch prediction hardware and how sophisticated prediction mechanisms can yield high accuracies. This hardware is used to generate profile statistics during program execution. After the program has completed execution and generated profile statistics, these statistics are then used by the compiler. When the same program is recompiled, better code optimization can be performed for subsequent program runs.

Conte's subject matter is distinctly different from the claimed inventions. Specifically, Conte's invention has nothing to do with instruction emulation. The claimed embodiments relate to speeding the execution of an original program under emulation within a single program run. Conte's invention does no emulation; it is about gathering profile information from one run, which can then be used by the compiler to generate better code for the same instruction set and the same machine.

In section 13, the Examiner rejected claims 8, 16 and 21 under 35 USC § 103 as obvious over Altman in view of Wall. Claims 16 and 21 have been canceled. Accordingly, this rejection is applied only to claim 8. Wall's report is entirely about LINK-time optimization. It is completely before the program is even generated/built. The program, therefore, has not even

run/executed (since it is yet to be built). Again, the claimed embodiment relates to execution time optimization.

It is common for large computer programs to have multiple source files that are all compiled separately. A compiler may not have enough knowledge about other code segments to be linked in later while compiling any single file. For this reason, a compiler has to be conservative in code generation including register allocation – the subject of Wall's report. Wall describes how parts of register allocation can be performed by the linker (rather than the compiler) when all parts of the program area available. This field is completely and distinctly different from the claimed invention, which relates to run-time and is unaffected by any link-time register allocation.

As stated above, at link-time, there may not be enough information about the actual hardware platform on which the program is going to execute to make such decisions. Without knowledge about the actual number of registers present in the system where the program is going to run, link-time register allocation is impossible in Wall.

In section 14, the Examiner rejected claim 9 under 35 USC § 103 as obvious over Altman in view of Conte. Conte uses branch prediction hardware to generate profiling statistics. The new profile statistics may be used by the compiler to generate different static branch prediction likelihoods for subsequent compilations and produce better optimized executables. This is different from the hardware using dynamic branch information to change the emulation sequence. In any case, Conte does not show anywhere the branch predictions of a group differing from individual branch predictions.

In section 15, the Examiner rejected claim 23 under 35 USC § 103 as obvious over Altman in view of common knowledge in the art. Claim 23 has been canceled.

In section 16, the Examiner rejected claim 24 under 35 USC § 103 as obvious over Altman in view of common knowledge in the art. As stated above, Altman is also a software approach with the limitations listed earlier.

In section 17, the Examiner rejected claim 25 under 35 USC § 103 as obvious over Altman in view of common knowledge in the art and further in view of Conte. Altman and Conte could be used to improve branch prediction. However, it should be noted that Conte's method involves gathering of profile information during a prior run. So, this can be used in conjunction with Altman's art that deal with static translation, not any of the "Dynamic Optimization". In any case, Altman's art is also a software approach with the limitations listed earlier.

In section 18, the Examiner rejected claims 26 and 27 under 35 USC § 103 as obvious over Altman in view of common knowledge in the art. As stated earlier, Altman is a software approach with the limitations listed earlier.

In section 19, the Examiner rejected claims 28 and 29 under 35 USC § 103 as obvious over Altman in view of common knowledge in the art and further in view of Wall. As mentioned earlier, Wall is about register allocation at link-time. Information such as lifetimes of registers that is available at link-time is not available at run-time; so Wall is not applicable. Further, actual register use is known only at run-time; so Wall is inherently limited in scope.

The Examiner's argument shows the limitations and ineffectiveness of Wall's approach. Wall needs to know well ahead, at link-time, specific hardware information of the target hardware platform, in this case, the number of registers. The claimed invention does not need this, as the hardware platform itself has the information. For example, Wall needs to know if the target system has 16 registers or 32 registers so that it can perform optimization accordingly.

Determining an algorithm to cycle through the registers is therefore not possible unless the hardware platform is fixed. Our invention has no such limitation.

In section 20, the Examiner rejected claim 30 under 35 USC § 103 as obvious over Altman in view of common knowledge in the art and further in view of Wall.

Wall's art does not have dynamic run-time information about how many times a particular section of code was visited and, therefore, the usage of specific registers. This is known only at run-time. So, Wall's register allocation cannot specify registers based on usage. Altman's dynamic optimization cannot be combined with Wall's technology which is link-time.

The Examiner's argument shows another of the limitations of Wall's approach. Wall has to use profile information which is from a previous run using (possibly) a different data set. If the input data or execution conditions change, the profile information is not valid and Wall's art becomes useless.

Claim 30 specifies "responding and changing, modifying the emulated sequence of instructions in response to at least the historical register usage information." This usage information is not known at link-time, only at run-time. For this reason, Wall's art is irrelevant.

In section 21, the Examiner rejected claim 31 under 35 USC § 103 as obvious over Altman in view of common knowledge in the art and further in view of Conte. As mentioned earlier, profile driven information (in Conte's case, provided by branch handling hardware) is the output for a particular program run with a specific data set. This is then used to generate prediction for all subsequent program runs with different input data. This approach is inherently limited; the claimed embodiments are far superior with the ability the change parameters based on current program execution.

If the Examiner has any questions or needs any additional information, the Examiner is invited to contact the undersigned.

Respectfully submitted,

By ____________________
Marc Sockol
Attorney for Applicant
Reg. No. 40,823

Dated: September 26, 2005
Squire, Sanders & Dempsey L.L.P.
600 Hansen Way
Palo Alto, CA 94304-1043
Telephone (650) 856-6500
Facsimile (650) 843-8777

CERTIFICATE OF MAILING

I hereby certify that this paper (along with any paper referred to as being attached or enclosed) is being deposited with the United States Postal Service on the date shown below with sufficient postage as first class mail in an envelope addressed to Mail Stop Amendment, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on
Date: September 26, 2005 By: ____________________
Ines Francetic

PaloAlto/87255.1